1. **What is a spring?**

   Spring is light weight and open source framework provides comprehensive infrastructure support for developing robust Java applications very easily and very rapidly.

2. **Explain spring framework History?**

   - ✓ Spring framework was initially written by Rod Johnson and was first released under the Apache 2.0 license in June 2003. Currently it's maintained by pivotal.
   - ✓ Spring initial name is Interface 21. Later it's modified to spring.
   - ✓ Spring introduced alternative to EJB.

3. **What is the latest version of spring and explain few features?**

   Latest spring version in 2021 is 5.3.4.

   New Features of spring 5
   ****

   - ✓ **JDK baseline update:** We must require Java 8 and above version.
   - ✓ **Core framework revision:** @Nullable and @NotNull annotations will explicitly mark nullable arguments and return values. This enables dealing null values at compile time rather than throwing NullPointerException at runtime.
   - ✓ Functional Programming With Kotlin
   - ✓ **Testing Improvements:** JUnit 5 Jupiter support started.
   - ✓ **Library Support:** JDBC 4, Hibernate 5 etc.

4. **Why spring is lightweight framework?**

   Spring is light weight because it won't depend on any Appserver and we can run the app with the help of spring library and jdk.

   The Spring Framework is very lightweight with respect to its size and functionality. This is due to its POJO implementation, which doesn't force it to inherit any class or implement any interfaces.

   Spring calls itself lightweight because you don't need all of spring to use part of it. For example, you can use Spring JDBC without Spring MVC.

5. **What's meant by loose & tight coupling?**

   Tight coupling means classes and objects are dependent on one another.

   Loose coupling means reducing the dependencies of a class that uses the different class directly.

Example:

Spring is loosely coupled. For example in India we have GSM (Ex Airtel if we are not interested in Airtel we can port it and move to any network it means loosely coupling.) and CDMA (Specific to that network only it means tightly coupling)

6. **What is meant by framework and explain it types?**

Framework, is a platform for developing software applications. It provides a foundation on which software developers can build programs for a specific platform.

Web Frame Work: It's for only web application development. Ex: Struts, JSF
Application Frame Work: It's for developing the Web as well as enterprise apps. Ex: Spring

7. **Differences between EJB and spring?**

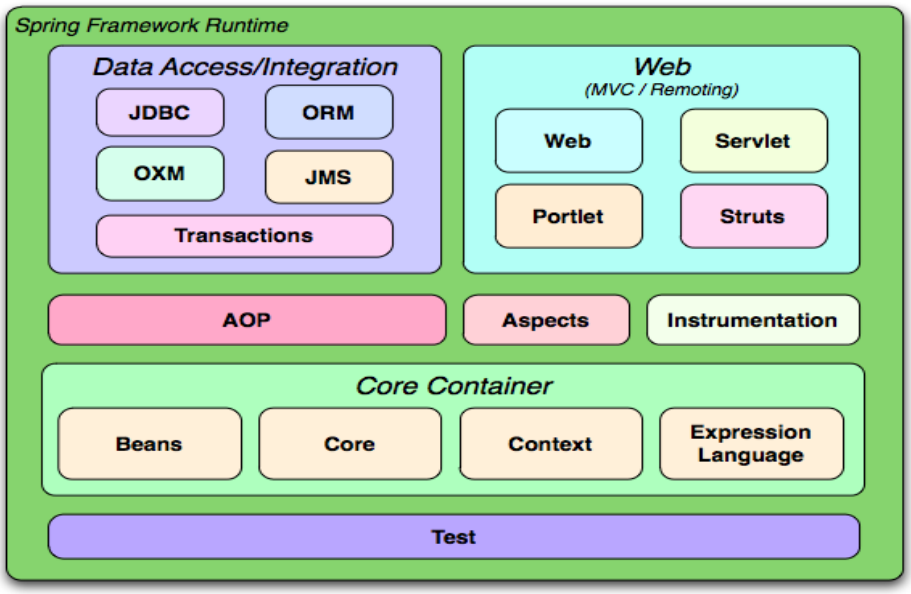
EJB	**spring**
Its heavy weight.	Its light weight
Its depends on Appserver	It's not required Server
It can inject anything in the container including EJB Data sources, JMS Resources, and JPA Resources.	It can inject anything including list, properties, map, and JNDI resources.
It tightly integrated to JPA.	It supports various persistence technologies such as JDBC, Hibernate, JPA, and iBatis.
EJB is a specification. The specification is implemented by many application servers such as GlassFish, IBM WebSphere and JBoss/WildFly. This means that our choice to use the EJB model for our application's backend development is not enough. We also need to choose which application server to use.	Spring is a framework.

8. **Explain the Advantages of spring?**

- ✓ **Loose Coupling:** The Spring applications are loosely coupled because of dependency injection.
- ✓ **Easy to test:** The Dependency Injection makes easier to test the application. The EJB or Struts application require server to run the application but Spring framework doesn't require server.
- ✓ **Lightweight:** Spring framework is lightweight because of its POJO implementation. The Spring Framework doesn't force the programmer to inherit any class or implement any interface. That is why it is said non-invasive.
- ✓ **Fast Development:** The Dependency Injection feature of Spring Framework and it support to various frameworks makes the easy development of JavaEE application.

- ✓ **Easy integration:** Easy to integrate any modules.
- ✓ **Predefined Templates:** Spring framework provides templates for JDBC, Hibernate, JPA etc. technologies. So there is no need to write too much code. It hides the basic steps of these technologies.

9. Explain the modules of spring?



Core: This is the heart of the spring framework. The Spring Core container contains core, beans, context and expression language (EL) modules.
Web: This group comprises of Web, Web-Servlet, Web-Struts and Web-Portlet. These modules provide support to create web application.
Data Access/Integration: This group comprises of JDBC, ORM, OXM, JMS and Transaction modules. These modules basically provide support to interact with the database.
AOP module: provides an aspect-oriented programming implementation allowing you to define method-interceptors and point cuts to cleanly decouple code that implements functionality that should be separated.
Test: module supports the testing of spring components with JUnit or TestNG frameworks.

10. Explain about spring core module?

- ✓ This module is used to develop the Standard applications.
- ✓ This module will provide us a way of implementing the Dependency Injection (DI) in spring frame work.
- ✓ Spring core module provides a spring container.
- ✓ This core module will provides us the way of implementing spring bean life cycle operations.

Spring core module having below parts:

---

1. Core
2. Bean
3. Context
4. Expression Language

Core: These modules provide IOC and Dependency Injection features.

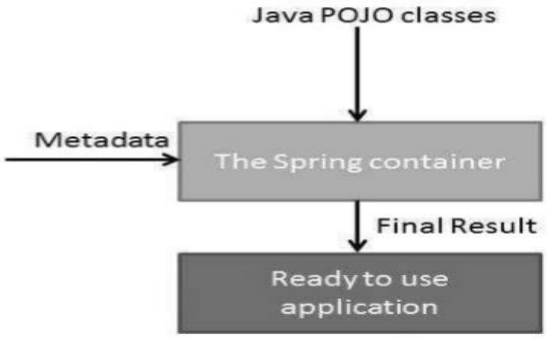
Bean: The Bean module provides BeanFactory, which is a sophisticated implementation of the factory pattern

Context: This module supports internationalization (I18N), EJB, JMS, Basic Remoting.

Expression Language: It is an extension to the EL defined in JSP. It provides support to setting and getting property values, method invocation, accessing collections and indexers, named variables, logical and arithmetic operators, and retrieval of objects by name etc.

11. Explain IOC Container?

The Spring container is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction. The spring container uses DI to manage the components that make up an application.
The container gets its instructions on what objects to instantiate, configure, and assemble by reading the configuration metadata provided. The configuration metadata can be represented either by XML, Java annotations, or Java code. The following diagram represents a high-level view of how spring works. The Spring IoC container makes use of Java POJO classes and configuration metadata to produce a fully configured and executable system or application.



12. Explain different types of IOC Container?

We have two types of ICO Container,

- BeanFactory

- ApplicationContext

Both BeanFactory and ApplicationContext provides a way to get a bean from Spring IOC container by calling getBean("bean name")

BeanFactory: A BeanFactory is like a factory class that contains a collection of beans. The BeanFactory holds Bean Definitions of multiple beans within itself and then instantiates the bean whenever asked for by clients.

The BeanFactory is the actual container which instantiates, configures, and manages a number of beans. These beans typically collaborate with one another, and thus have dependencies between themselves. These dependencies are reflected in the configuration data used by the BeanFactory.

BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

Usually, the implementations use lazy loading, which means that Beans are only instantiating when we directly calling them through the getBean() method.

Example:

```
XmlBeanFactory factory = new XmlBeanFactory (new ClassPathResource("beans.xml"));
HelloWorld obj = (HelloWorld) factory.getBean("helloWorld");
obj.getMessage();
```

ApplicationContext: The ApplicationContext is the central interface within a spring application that is used for providing configuration information to the application.

It implements the BeanFactory interface. Hence, the ApplicationContext includes all functionality of the BeanFactory and much more! Its main function is to support the creation of big business applications.

This container adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners.

```
Exmple::
---------------
ApplicationContext context=new ClassPathXmlApplicationContext("beans.xml");
HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
obj.getMessage();
```

13. **Difference between BeanFactory and ApplicationContext?**

BeanFactory	ApplicationContext
Loads beans on-demand	Loads all beans at startup
It support only two scopes (singleton & prototype)	It will support singleton, request, prototype and session.
This is the base interface	ApplicationContext interface is built on top of the BeanFactory interface.
It does not support Internationalization.	It support Internationalization.
It does not support annotation based dependency	It will support annotation based dependency
It will useful for lightweight applications like Mobile and applets.	It will support all kinds of applications.

14. **Explain different types of BeanFactory and ApplicationContext?**

    The most commonly used BeanFactory implementation,

    XmlBeanFactory: This container reads the configuration metadata from an XML file and uses it to create a fully configured system or application.

    The most commonly used ApplicationContext implementation,

    FileSystemXmlApplicationContext – This container loads the definitions of the beans from an XML file. Here you need to provide the full path of the XML bean configuration file to the constructor.

    ClassPathXmlApplicationContext – This container loads the definitions of the beans from an XML file. Here you do not need to provide the full path of the XML file but you need to set CLASSPATH properly because this container will look bean configuration XML file in CLASSPATH.

    WebXmlApplicationContext – This container loads the XML file with definitions of all beans from within a web application.

15. **What is meant by Dependency Injection?**

    Dependency Injection is a process of removing the dependency from programming code so that it's easy to manage and test the application. Dependency Injection makes our programming code loosely coupled.

16. **Explain different types of Dependency Injection?**

    There are three types of dependency injection,

    - Constructor Based Injection
    - Setter Based Injection
    - Field Based Injection

17. **What is meant by Constructor Based Injection what kind of values we can insert via constructor dependency injection?**

The process of injecting the dependencies through constructor is called as Constructor Based Injection.

We can insert below kind of values via constructor based injection,

- ✓ Primitive and String-based values
- ✓ Dependent object (contained object)
- ✓ Collection values etc.

Imp: Primitive/String based values injection is possible in XML based configuration only. This was not possible with @Autowired annotation.

```
private int id;
private String name;
private List<String> address;
private AddressBean addressBean;

@Autowired
public StudentBean(int id, String name,List<String> address,AddressBean addressBean) {
    this.id=id;
    this.name=name;
    this.address=address;
    this.addressBean=addressBean;
}
```

18. **What is meant by Setter Based Injection what kind of values we can insert via Setter dependency injection?**

The process of injecting the dependencies through setter method is called as Setter Based Injection.

We can insert the below values in Constructor Based Injection,

- ➢ Primitive and String-based values
- ➢ Dependent object (contained object)
- ➢ Collection values etc.

```java
public class AddressBean {
    private int addressId;
    private String city;

    @Autowired
    public void setAddressId(int addressId) {
        this.addressId = addressId;
    }

    @Autowired
    public void setCity(String city) {
        this.city = city;
    }

    public int getAddressId() {
        return addressId;
    }

    public String getCity() {
        return city;
    }
```

19. What is meant by Field Based Injection?

This type of injection is possible only in the annotation based approach.

In this type of Dependency Injection, spring assigns the dependencies directly to the fields. It is different than Constructor Injection or Setter based Dependency Injection.

The interesting thing to understand is, spring injects the dependencies, even if the field is private. Spring uses Java Reflections to do so. Hence it is called unsafe by many of the experts.

```
Example:
**********
@Service
class DependentService {
    @Autowired
    private Service1 service1;
    @Autowired
    private Service2 service2;

    void doSmth() {
        service1.doSmth();
        service2.doSmth();
    }
}
```

20. Explain advantages and disadvantages of Field based Injection?

Advantages:
****

- ✓ Easy to use, no constructors or setters required
- ✓ Can be easily combined with the constructor and/or setter approach

Disadvantages
****

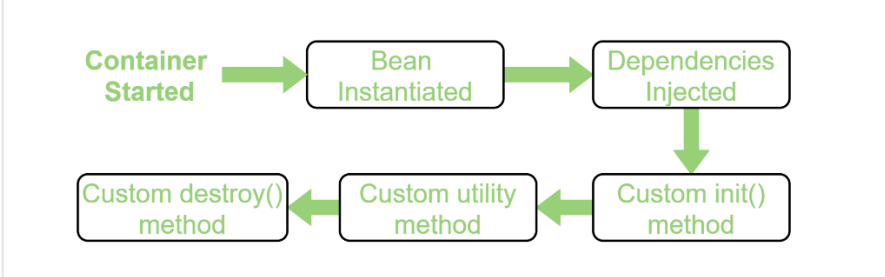
- ✓ Less control over object instantiation. In order to instantiate the object of a class for a test, you will need either a spring container configured or mock library — depends on the test you are writing.
- ✓ A number of dependencies can reach dozens until you notice that something went wrong in your design.
- ✓ No immutability — the same as for setter injection.
- ✓ As there is no constructor/Setter methods it may create the same object twice.

21. Difference between Field based, Setter based and Constructor based Injection?

Constructor Based	Setter Based	Field Based
Supports immutability.	No Immutability	No Immutability
State Safe. The object is instantiated to a full state or is not instantiated at all.	Consumer uses no-argument constructor. And possibility miss calling one of the setters or call same setter twice with different value (copy-paste bugs)	Consumer uses no-argument constructor. There is no valid way to set state of the object. Only option is to use Reflection to set the private fields.
It's not good when we have less number of fields	It's not good when we have more number of fields	It's not good when we have more number of fields.
Partial dependency Injection is possible like if we have 3 dependencies like int, String and Long if we pass only two it will take the default value for third one.	It will not support partial injection we should inject all fields	

22. Explain about spring lifecycle?

The lifecycle of any object means when & how it is born, how it behaves throughout its life, and when & how it dies.
Bean life cycle is managed by the spring container. When we run the program then, first of all, the spring container gets started. After that, the container creates the instance of a bean as per the request and then dependencies are injected. And finally, the bean is destroyed when the spring container is closed.
if we want to execute some code on the bean instantiation and just after closing the spring container, then we can write that code inside the custom init() method and the destroy() method.



23. Explain the xml and annotations based way to invoke the custom init and destroy methods?

XML based way:

In this approach, in order to avail custom init() and destroy() method for a bean we have to register these two methods inside Spring XML configuration file while defining a bean.

Therefore, the following steps are followed:
Firstly, we need to create a bean HelloWorld.java in this case and write the init() and destroy() methods in the class.

```
public class HelloWorld {
    // This method executes
    // automatically as the bean
    // is instantiated
    public void init() throws Exception
    {
        System.out.println(
            "Bean HelloWorld has been "
            + "instantiated and I'm "
            + "the init() method");
    }

    // This method executes
    // when the spring container
    // is closed
    public void destroy() throws Exception
    {
        System.out.println(
            "Conatiner has been closed "
            + "and I'm the destroy() method");
    }
```

```
<beans>
     <bean id="hw" class="beans.HelloWorld"
              init-method="init" destroy-method="destroy"/>

</beans>
```

Annotation based way:

To provide the facility to the created bean to invoke custom init() method on the startup of a spring container and to invoke the custom destroy() method on closing the container, we need annotate init() method by @PostConstruct annotation and destroy() method by @PreDestroy annotation.

```
public class HelloWorld {
     // Annotate this method to execute it
    // automatically as the bean is
    // instantiated
    @PostConstruct
    public void init() throws Exception
    {
        System.out.println(
            "Bean HelloWorld has been "
            + "instantiated and I'm the "
            + "init() method");
    }

    // Annotate this method to execute it
    // when Spring container is closed
    @PreDestroy
    public void destroy() throws Exception
    {
        System.out.println(
            "Conatiner has been closed "
            + "and I'm the destroy() method");
    }
```

24. What is meant by Spring Bean explain beans scopes?

The beans which are created with the help of configuration metadata that you supply to the container is called as spring beans.

We have below scopes available for spring beans,

Singleton: This scopes the bean definition to a single instance per Spring IoC container (default). Will use this scope for **stateless** beans. While using this scope, make sure bean doesn't have shared instance variables otherwise it might lead to data inconsistency issues.

Prototype: This scopes a single bean definition to have any number of object instances. Will use this scope for **statefull** beans.

Request: This is same as prototype scope, however it's meant to be used for web applications. A new instance of the bean will be created for each HTTP request.

Session: A new bean will be created for each HTTP session by the container.

application: In application scope, container creates one instance per web application runtime.

25. Difference between Singleton and Application scope?

application scoped bean is singleton per ServletContext, whereas singleton scoped bean is singleton per ApplicationContext. Please note that there can be multiple application contexts for single application.

A ServletContext is shared between all servlets living on the same servlet container (e.g. Tomcat). This is a Java EE class (it belongs to the package javax.servlet). Beans annotated with @ApplicationScope are bounded to the ServletContext.

An ApplicationContext represents a Spring IoC container, so it is a Spring-specific class. Singleton scoped beans are bounded to the ApplicationContext.

26. Give an example to define the beans scopes in XML and Annotation based way?

Xml Based:

```
<bean id = "hw" class = "bean.HelloWorld" scope = "singleton"/>
<bean id = "hw" class = "bean.HelloWorld" scope = "prototype"/>
<bean id = "hw" class = "bean.HelloWorld." scope = "request"/>
<bean id = "hw" class = "bean.HelloWorld" scope = "session"/>
<bean id = "hw" class = "bean.HelloWorld" scope = "application"/>
```

Annotation Based:

```
@Bean(name = "singletonBean")
@Scope("singleton")
public SingletonBean getSingletonBean() {
    return new SingletonBean();
}

@Bean(name = "protoTypeBean")
@Scope("prototype")
public ProtoTypeBean getProtoTypeBean() {
    return new ProtoTypeBean();
}

@Bean(name = "requestBean")
@RequestScope
public RequestBean getRequestBean() {
    return new RequestBean();
}

@Bean(name = "sessionBean")
@SessionScope
public SessionBean getSessionBean() {
    return new SessionBean();
}

@Bean(name = "applicationBean")
@ApplicationScope
public ApplicationBean getApplicationBean() {
    return new ApplicationBean();
}
```

27. Explain about POJO in spring?

POJO in Java stands for Plain Old Java Object. It is an ordinary object. POJO class contains variables and their Getters and Setters. The POJO file does not require any special classpath. It increases the readability & re-usability of a Java program. It will not extend any predefined classes or interfaces.

28. Difference between POJO and Java Bean?

As we know that in Java POJO refers to the Plain old Java object. POJO and Bean class in Java shares some common features which are as follows −

- ✓ Both classes must be public i.e accessible to all.
- ✓ Properties or variables defined in both classes must be private i.e. can't be accessed directly.
- ✓ Both classes must have default constructor i.e no argument constructor.
- ✓ Public Getter and Setter must be present in both the classes in order to access the variables/properties.

The only difference between both the classes is Java make java beans objects serialized so that the state of a bean class could be preserved in case required. So due to this a Java Bean class must either implements Serializable or Externalizable interface.

29. What is meant by Autowiring?

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.

Autowiring can't be used to inject primitive and string values. It works with reference only.

30. Explain different types of Autowiring?

No: This option is default for spring framework and it means that autowiring is OFF. You have to explicitly set the dependencies using <property> tags in bean definitions.

byname: This option enables the dependency injection based on bean names. When autowiring a property in bean, property name is used for searching a matching bean definition in configuration file. If such bean is found, it is injected in property. If no such bean is found, a error is raised.

byType: This option enables the dependency injection based on bean types. When autowiring a property in bean, property's class type is used for searching a matching bean definition in configuration file. If such bean is found, it is injected in property. If no such bean is found, a error is raised.

constructor: Autowiring by constructor is similar to byType, but applies to constructor arguments. In autowire enabled bean, it will look for class type of constructor arguments, and then do a autowire bytype on all constructor arguments. Please note that if there isn't exactly one bean of the constructor argument type in the container, a fatal error is raised.

Example for all types:

```java
private int id;
private String name;
private List<String> address;

@Autowired //Autowired byName
private AddressBean addressBean;

@Autowired ////Autowired constuctor
public StudentBean(int id, String name,List<String> address,AddressBean addressBean) {
    this.id=id;
    this.name=name;
    this.address=address;
    this.addressBean=addressBean;
}

@Autowired //Autowire byType
public void setId(int id) {
    this.id = id;
}
```

31. Explain the advantages and disadvantages of Autowiring?

Advantages:

- ✓ It requires the less code because we don't need to write the code to inject the dependency explicitly.
- ✓ No need to go for XML configurations.

Disadvantages:

- ✓ Wiring information may not be available to tools that may generate documentation from a spring container.
- ✓ It can't be used for primitive and string values.
- ✓ **Overriding possibility:** You can still specify dependencies using <constructor-arg> and <property> settings which will always override autowiring.

32. Spring follows which design pattern?

- ✓ Inversion of Control/Dependency Injection
- ✓ Factory Pattern (Bean Factory/Application Context)
- ✓ Singleton Pattern
- ✓ Proxy Pattern (Proxy design pattern is widely used in AOP)
- ✓ Front Controller (Centralized request handling)
- ✓ Model-view-Controller
- ✓ View Pattern (Dealing with the Views like JSP/FTL)
- ✓ Template Pattern (Dealing with Spring JDBC, JPA)

33. Explain the beans tags of spring configuration xml file?

```
//Baisc Bean Config
<bean id="country" class="org.arpit.java2blog.model.Country">
</bean>

//Constructor Injection
<bean id="country" class="org.arpit.java2blog.model.Country">
      <constructor-arg index="0" value="India"></constructor-arg>
 </bean>

 //Setter Injection
 <bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld">
    <property name = "message" value = "Hello World!"/>
 </bean>

  //Scope
 <bean id = "hw" class = "bean.HelloWorld" scope = "singleton"/>

 //Manual init & destroy
 <bean id="country" class="org.arpit.java2blog.model.Country" init-method = "init" destroy-method = "destroy">
</bean>

//Default initialization and destroy methods
<beans xmlns = "http://www.springframework.org/schema/beans"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
  default-init-method = "init"
  default-destroy-method = "destroy">
  <bean id = "hw" class = "bean.HelloWorld" scope = "singleton"/>
  </beans>
```

34. Explain the use of default init and destroy methods?

If you have too many beans having initialization and/or destroy methods with the same name, you don't need to declare init-method and destroy-method on each individual bean. Instead, the framework provides the flexibility to configure such situation using default-init-method and default-destroy-method attributes on the <beans> element

```
<beans xmlns = "http://www.springframework.org/schema/beans"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
   default-init-method = "init"
   default-destroy-method = "destroy">

   <bean id = "..." class = "...">
      <!-- collaborators and configuration for this bean go here -->
   </bean>

</beans>
```